TAMPERE UNIVERSITY OF TECHNOLOGY

SYED ABDUL SAMAD

RANDOM WALK OVERSAMPLING TECHNIQUE FOR MINORITY CLASS CLASSIFICATION

Master of Science Thesis

Examiner: Prof. Tapio Elomaa

Examiners and topic approved in the council meeting of Faculty of Information Technology on March 6th, 2013.

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Information Technology
SYED ABDUL SAMAD: Random walk oversampling technique for minority class classification
Master of Science Thesis, 42 pages
April, 2013
Major: Software systems
Examiner: Prof. Tapio Elomaa
Keywords: Imbalance Datasets, Oversampling Technique, Random Walk, Markov Chain Monte Carlo, Gibbs Sampling

Learning classifiers from imbalanced or skewed datasets is an important topic, arising very often in practice in classification problems. In such problems, almost all the instances are labeled as one class, while very few instances are labeled as the other class, usually the more important class. Traditional classifiers trying to achieve an accurate performance over a full range of instances are not suitable to deal with imbalance learning tasks. They tend to classify all the data into the majority class, which is usually the less important class. Researchers have already presented many solutions to this problem both on data and algorithmic level.

In this thesis a new approach to deal with imbalanced datasets is presented on the data level. This approach is an oversampling technique which involves generating new samples for the minority class by making a random walk in the dataset. The new samples are generated by some Markov Chain Monte Carlo Algorithm. Newly generated samples are then added to existing data set in order to balance the ratio between majority and minority class samples.

PREFACE

This thesis is submitted in partial fulfillment of the requirements for a Masters Degree in Information Technology in Tampere University of Technology, Finland. The completion of this thesis also results in the completion of my Master degree. This thesis provides a new approach to deal with imbalance data classification problem in machine learning. I want to share my profound gratitude to my supervisor Prof. Tapio Elomaa, who provided me this research topic for my thesis and encouraged me with his help and his guidance along the way. I also want to thank my supervisor for giving me enough time for the meetings on weekly basis.

I am highly thankful to all my family specially my Mom and Dad for believing in me and always being a continuous support for me during the time i was away from my home. Special thanks to all my sisters and two little nieces (Isra and Eraj) for providing me all the love and support. I am also highly thankful to my grandmother for her continuous prayers for me.

In the end i want to thank all my friends in Tampere for giving me the great company. Mr. Ahmad Tariq for always being a support for me, special thanks to Ata-ul-Ghalib, Azaz Ahmad and Faraz Ahmad for helping me during difficult times in my thesis, Waheed Ahmad for being a continuous motivation for me. I am also grateful to Muhammad Ahsan, Masoom Ahmad and Khalid Latif for their guidance and help during my studies.

April 18, 2013

Syed Abdul Samad
Tampere, Finland
thesamad@gmail.com
+358-44-0466277

CONTENTS

LIST OF ABBREVIATIONS

CL Tree Chow-Liu Tree

CNN Condensed Nearest Neighbor

DAG Directed Acyclic Graph

EVS Evolutionary Prototype Selection

FDR False Detection Rate

FN False Negative

FP False Positive

HBP Hypothesis Boosting Problem

KNN K Nearest Neighbor

MC Markov Chain

MCMC Markov Chain Monte Carlo

MCS Minority Class Samples

MI Mutual Information

MST Minimal Spanning Tree

MWMOTE Majority Weighted Minority Oversampling TEchnique

NN Nearest Neighbor

NCR Neighborhood Cleaning Rule

ROC Receiver Operating Characteristics

AUC Area Under ROC

SMOTE Synthetic Minority Oversampling TEchnique

SVM Support Vector Machine

TN True Negative

TP True Positive

TR Training Set

UCI University of California Irvine

WEKA Waikato Environment for Knowledge Analysis

DEFINITIONS

Bayesian Network

A Bayesian network is a set of random variables and their conditional dependencies represented via a DAG.

Clustering

Clustering is the assignment of a set of observations into subsets (called clusters) so that observations within the same cluster are similar according to some predesignated criterion or criteria, while observations drawn from different clusters are dissimilar.

Cross-Validation

Cross-validation is a technique which is used in estimation, how accurately a predictive model will perform in practice. One round of cross-validation involves partitioning a sample of data into complementary subsets, performing the analysis on one subset, and validating the analysis on the other subset.

Feature Vector

In pattern recognition and machine learning, a feature vector is an n-dimensional vector of numerical features that represent some object.

Joint Probability

Joint probability is a measure of two or more events happening at the same time. Joint probability is the probability of event Y occurring at the same time event X occurs.

Marginal Probability

The probability of one variable taking a specific value irrespective of the values of the others.

Mutual Information

Mutual Information measures mutual dependence of two random variables on each other.

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

A dataset is said to be imbalanced when at least one class (*minority class*) has very few instances as compared to the other classes (*majority class*). Usually the instances of minority class are more important from the classification point of view and are called *positive instances* and the instances of the majority class are called *negative instances*. In machine learning and pattern classification, learning from imbalanced data has been under research in the recent years. The aim in learning from imbalanced data is to develop an automated approach that correctly predicts the minority class instances. The development of such an approach will have a direct impact on real life applications, such as medical diagnosis, fraudulent telephone calls detection [1], text classification [2], oil spills detection for satellite images [3], and so on, where it is more important to classify minority class instances correctly than the ones of majority class. For example, in real-world applications mispredicting a rare event can result in more serious consequences than mispredicting a common event. For example in the case of cancerous cell detection, misclassifying non-cancerous cells leads to additional clinical testing but misclassifying cancerous cells leads to very serious health risks.

It has been observed that traditional classifiers do not perform up to the mark when dealing with imbalanced data, as they assume that the target classes share similar prior probabilities. In this case, standard classifiers tend to be overwhelmed by the majority class and ignore the minority class. Importance of learning from imbalanced data grew as more and more researchers realized that this imbalance causes suboptimal classification performance, and that most algorithms behave badly when the datasets are highly imbalanced. Barandela et al. [4] described that the classifier accuracy, when dealing with imbalanced data cannot be measured as average classification accuracy for all the different classes in the dataset. For example, a dataset with 99% of negative instances and 1% of positive instances. In such a situation, if traditional classifier classifies all the dataset as negative instances then the overall accuracy of the classifier will be 99%, since it has classified 1% of positive instances incorrectly.

Guo et al. [5] described that from the application point of view, class imbalance problem falls in two categories: natural imbalance problems (like credit card frauds

and rare diseases) and those problems, in which the data is not naturally imbalanced, but it is very expensive to obtain examples of the minority class for learning and to to make classifiers for them. For example, in shuttle failure, it is very hard to obtain the data to develop some automated approach to handle these rare events. However, we should note that imbalance ratio between classes is not the only factor that reduces classification performance, other factors such as training size, dataset characteristics and concept complexity also affect performance.

Japkowics [6] presented two very important questions for dealing with the class imbance problem. The questions are: What types of imbalances hinder the accuracy performance of standard classifier? and What approaches for dealing with the class imbalance problem are most appropriate? These questions are important since their answers may suggest fruitful directions for future researchers focus their inquiry onto the particular type of solution found most promising, given the particular characteristics in their application domain.

A more difficult version of imbalance class problem is multi-class classification. In multi-class classification there are more than one minority classes and the task is to correctly classify all the minority class instances. Multi-class classification problem can always be converted into two class imbalance problem, by considering only one minority class at a time and treating all the other instances as majority class instances. However, Wang et al. [7] described that treating multi-class classification problem as a two class classification problem does not always end up in the required results, and proposed a new methodology which combines the oversampling technique with AdaBoost.NC [8] for dealing with multi-class classification and showed, that this approach results in much better classification results in multi-class classification problem.

In this thesis, we have developed a new approach to deal with imbalanced datasets, it is an oversampling technique which includes generating new samples for minority class. New samples for the minority class are generated by making a random walk using Gibbs sampling which belongs to Markov Chain Monte Carlo (MCMC) family of algorithms. We have compared our approach with the original dataset results and with some other oversampling techniques such as SMOTE.

This thesis is divided into 6 main chapters. Chapter 2, Literature Review, discusses the work that has already been done in solving class imbalance problem on both data and algorithmic level. Chapter 3, Proposed Approach, in detail discusses our approach to handle class imbalance problem. Chapter 4, Empirical Evaluation, describes the results of our approach and compares the results with original dataset and other oversampling approaches. Chapter 5, Further Improvements, discusses

any further improvements that can be done to improve our approach. Chapter 6, Conclusion, concludes the thesis with some discussion on our approach.

2. LITERATURE REVIEW

In this chapter we describe state of the art research that has already been done in learning from imbalanced data. We will explain different methods and techniques used for learning and more accurately classifying minority class from imbalanced datasets.

Learning from imbalanced datasets has already been addressed by many researchers on both *data* and *algorithmic* level. On the algorithmic (*internal*) level, idea is to modify existing algorithms, to make them perform better with imbalanced datasets without changing the actual data. Techniques such as, recognition-based approach, cost-sensitive learning, bagging and boosting are used. Internal approaches in some cases are very effective and their results are much better than those of external approaches but internal approaches have the problem of being too much data specific because of the differences of the data characteristics. Hence it is very difficult to create a general classifier that deals with all the different datasets.

Unlike internal approach, the main idea on data (*external*) level is to adjust majority and minority class samples ratio. This is usually done by data re-sampling, which include both over- and undersampling of data. The aim is to adjust the data ratio in such a way that the minority class will no longer remain heavily unrepresented among the total data data under consideration. Techniques such as Synthetic Minority Oversampling TEchnique (**SMOTE**), Borderline SMOTE, Random Over- and Undersampling, Neighborhood cleaning rule and Evolutionary Prototype Selection are some examples for handling imbalanced datasets on data level.

Recently Garcia et al. [9] described that, research lines within the general framework of class imbalance have been divided in the following groups:

- Re-sampling methods for balancing the dataset.
- Modification of existing learning algorithms.
- Measuring the classifier performance in imbalance domains.
- Relationship between class imbalance and other data complexity characteristics.

We will describe both the data and algorithmic level techniques and their effects in the following sections.

2.1 Data Level Techniques

Data level technique includes resampling of the original dataset. Data resampling includes both over- and undersampling of the data. Japkowicz et al. [10] presented three most important questions that should be considered while resampling data.

- Should we oversample or undersample?
- At what rate should this oversampling and undersampling take place?
- Can a combination of different expressions of resampling paradigm help improve classification accuracy?

2.1.1 Oversampling Techniques

Random Oversampling

Random Oversampling is a very naive approach to data oversampling. It simply replicates the minority class examples and adds them to the training data. By using this technique new examples come from the existing minority class examples in the training set (TR). This results in the problem of over-fitting.

Over-fitting is a problem that occurs when all the training examples are very similar to each other, and these examples are correctly classified by the classifier. In such a scenario if a test example is slightly different from the training examples then the classifier is not able to classify it correctly and results in poor classification for the new examples. In other words classifier is trained to classify only very narrow set of examples correctly.

Synthetic Minority Oversampling TEchnique (SMOTE)

SMOTE is an oversampling technique first introduced by Chawla et al. [11]. They describe a new technique for oversampling of data by creating *synthetic* examples instead of generating new examples with replacement. New synthetic examples are

introduced in the minority class by using K nearest neighbor *(KNN)* rule . A nearest neighbor *(NN)* rule [12] classify an unclassified sample same as its *NNs*, when the neighboring samples are already correctly classified by some external source earlier. The choice of K is also something that needs to be addressed. Usually K is taken as **3, 5** or **7** depending on the data. In SMOTE synthetic examples are introduced along the respective line segments of all the K minority class *NN*. It results the decision regions to be more towards the majority class and less specific. New synthetic examples are generated in the following way:

Take the differences between the feature vector (sample) and its NNs *under consideration. Multiply these differences by random numbers between 0 and 1, and add them to the feature vector under consideration.*

As the feature vector consists of many different values, the difference between the two feature vectors can be computed by taking the difference between the corresponding values of the feature vector.

Border-Line SMOTE

Border-Line SMOTE technique consists of Border-Line SMOTE1 and Border-Line SMOTE2. Han et al. [13] gave emphasis on those examples in minority class that are close to decision boundary of majority and minority class. A decision boundary can be defined as a boundary which separates two or more classes from each other in a dataset. Han et al. think that the examples which are on the border-line are more apt to be misclassified as compared to the examples that are farther away from the decision boundary.

In Border-Line SMOTE the first step is to isolate borderline minority class examples by calculating *KNNs*, and put them in another set. These separated examples are considered to be in danger of being misclassified, and hence this set is called **DANGER** set. Then SMOTE is applied to the examples in the DANGER set in order to generate new examples. In Border-Line SMOTE1 synthetic examples are generated only by considering the minority class in the original distribution whereas in Border-Line SMOTE2 synthetic examples are also generated from the majority class examples instead of considering minority class only.

Safe-Level SMOTE

Bunkhumpornpat et al. [14] recently introduced a new technique of oversampling similar to SMOTE called Safe-Level SMOTE. Bunkhumpornpat et al. showed that by using Safe-Level SMOTE technique they have achieved better accuracy performance than SMOTE and Border-Line SMOTE. Safe-Level can be defined as follows:

safe-level (sl) = **the number of positive instances in *KNNs***

and safe-level ratio can be defined as:

safe-level ratio = **sl of a positive instance / sl of *KNNs***

Safe-Level SMOTE works by assigning safe-level to all the positive examples before generating synthetic examples. However, safe-level should be computed for all the samples in dataset. In order to generate new examples in safe region all the synthetic examples are placed close to the largest safe-level. Safe-level ratio is used for selecting safe position to create new synthetic examples. We need to compute the safe-level of *KNNs* for calculating safe-level ratio of a sample, this can be done by computing the safe-level for all the KNNs of a sample for which we are calculating the safe-level ratio. An example is considered as noise if its safe-level is close to 0, and considered as safe if its safe-level is close to K.

Majority Weighted Minority Oversampling TEchnique

Barua et al. [15] recently described, that most of the synthetic oversampling methods, in some scenarios generate wrong synthetic examples, which makes it hard for the classifier to classify new examples correctly. They introduced a new method of data oversampling called Majority Weighted Minority Oversampling TEchnique for Imbalanced Dataset Learning (MWMOTE).

MWMOTE works by first identifying examples in the minority class that are difficult to learn, it then assigns them different weights depending on the learning difficulty and makes their cluster inside minority class. It then generates synthetic examples from these examples and adds them to the minority class. Barua et al. claim that this method results in better classification results than other oversampling methods.

2.1.2 Undersampling Techniques

Random Undersampling

The idea of random undersampling is contrary to random oversampling described in Section 2.1.1. In random undersampling we randomly remove example from majority class, in order to balance the class instances. This results in the removal of very important information from the majority class. This approach also results in downsizing of the training data considerably. Therefore it is the most naive approach in data undersampling.

Neighborhood Cleaning Rule

Neighborhood Cleaning Rule (NCR) was first introduced by Laurikkala [16]. The main idea in NCR is data cleaning instead of reducing the size of data by removing useful information from the majority class. Laurikkala described that, the size of the class is not important in correct classification as opposed to the examples in the class, which are important in the classification. Hence data reduction is not a good option in undersampling.

NCR works by identifying the borderline examples in the majority class. Borderline examples are identified by *KNN* rule. A borderline example is considered noisy if the class label is different from its 3 neighboring examples. If the borderline example e_i under consideration is a majority class example and its two or more neighbors are from minority class then this example is considered as noise and is then removed from the dataset. If e_i is a minority class example and its *NNs* are majority class examples then *NNs* are removed from the data.

Tomek Links

Tomek Links results in the removal of noise or borderline majority class examples. It is another undersampling method first introduced by Tomek [17]. It works by identifying noise and borderline examples on the decision boundary.

Consider two examples x_i and x_j belong to different classes. Let d(x_i,x_j) be the distance between them. Examples (x_i,x_j) form a tomek link if there is no other example x_l ($l \neq i$ and $l \neq j$), such that d(x_i, x_l) <d(x_i, x_j) and d(x_j, x_l) <d(x_i, x_j). If x_i and x_j create a tomek link, then either one of them is noise or both are borderline examples. Originally tomek links are used to find out noise and borderline

examples, but this technique is also used as an undersampling method for majority class examples.

Evolutionary Prototype Selection

Garcia et al. [18] proposed Evolutionary Prototype Selection (EVS) method, which makes use of genetic algorithm. The main idea of EVS is to keep best examples from majority class and remove redundant and borderline examples. EVS uses a fitness function to compute the fitness of majority class example. Let S be a subset data samples from TR. Fitness function can be defined as a combination of two values, classification rate ($clas\_rat$) associated with S and percentage of reduction ($perc\_red$) of instances of S with regard to TR.

$$Fitness(S) = \alpha \times clas\_rat + (1 - \alpha) \times perc\_red$$

and the percentage of reduction is :

$$perc\_red = 100 \times \frac{|TR| - |S|}{|TR|}$$

EVS results in moving the decision boundary more towards the majority class. It also prevents the overfitting problem for minority class.

Condensed Nearest Neighbor Rule

Condensed Nearest Neighbor (CNN) [19] rule is an improved version of *NN* rule with less memory requirements. CNN works by creating a consistent subset which is later used in classification of new samples. It is also used as an undersampling method for majority class, as the samples which are not consistent are thrown away. A consistent subset of original data classifies all the samples correctly, when used as a reference for new samples.

The samples are stored in two bins, called STORE and GRABBAG. First sample is picked up randomly and stored in STORE. Next sample is picked and by using *NN* rule, it is classified using STORE as reference. If the new sample is classified correctly then it is placed in GRABBAG and is put in STORE otherwise. Similarly all the samples in the original datasets are classified and either kept in STORE or in GRABBAG. After one complete pass through the original data, samples that are kept in GRABBAG are classified until GRABBAG becomes empty or no sample is

transferred from GRABBAG to STORE.

After completing the above procedure GRABBAG is thrown away and STORE is the consistent subset. Hence redundant examples in the original dataset are removed from the original dataset.

Cluster Based Undersampling Approaches

Yen and Lee [20] described an undersampling method for imbalanced datasets which is based on clustering. In this method all the data is divided into K clusters. Each cluster has its distinct characteristics, which depends on the majority and minority class examples in a cluster. Let number of majority and minority class examples in the original data be S_{MA} and S_{MI}. If a cluster has more majority class examples than minority class examples then it will behave as majority class cluster and vice versa.

Let the numbers of the majority class examples in i^{th} cluster be much greater than the number of the minority class examples. The majority and minority class examples in i^{th} cluster be S^i_{MA} and S^i_{MI}. Then, the ratio of majority class examples to minority class examples will be S^i_{MA}/S^i_{MI}. As the number of majority class examples in a cluster is much more than the minority class examples, therefore, the number of selected majority class examples in i^{th} cluster will be:

$$SS^i_{MA} = m \times S_{MI} \times \frac{S^i_{MA}/S^i_{MI}}{\sum_{i=1}^{K} S^i_{MA}/S^i_{MI}}$$

More majority class examples are selected from the cluster which behaves more like majority class cluster for the final undersampled data. Hence, the contribution of SS^i_{MA} is more in the final dataset if it has more majority class examples than minority class examples. The majority class examples are randomly selected from each cluster after determining the majority class examples that are to be selected from i^{th} cluster. Approximately $m \times S_{MI}$ majority class examples are selected. These examples are then merged with minority class examples, to get new balanced dataset.

We have described many over- and undersampling methods above. The overall objective of the above methods is to improve the classification rate of minority class in imbalanced datasets. Oversampling methods achieve this by increasing the number of minority class samples, while undersampling methods do this by removing the majority class samples. One can always use a combination of both these methods,

for example He et al. [21] have shown that the combination of SMOTE and Tomek Links results in much more balanced dataset than the original dataset.

2.2 Algorithm Level Techniques

Algorithm level techniques involve modification in the algorithm according to the dataset characteristics. Algorithm level results in more accurate results than data level techniques.

2.2.1 Cost Sensitive Learning

Cost Sensitive Learning is an algorithmic level method, which takes misclassification cost into consideration when classifying data samples. It targets to minimize the overall misclassification cost in order to correctly classify given data. Misclassification cost may vary from class to class in the data. In imbalanced data the misclassification cost of minority class examples must be greater than that of majority class examples. Given a specification of correct and incorrect prediction, classification of an example should lead to the lowest possible cost. Misclassification cost is specified usually in cost matrix. Cost matrix specify the cost of classification when the sample actually belongs to class j and classifier classifies it in class i [22,23].

Table 2.1: Cost Matrix.

	Actual Negative	Actual positive
Predict Negative	C(0,0) = c_{00}	C(0,1) = c_{01}
Predict Positive	C(1,0) = c_{10}	C(1,1) = c_{11}

Let C be the cost matrix (see Table 2.1), specifying the misclassification cost, when an example belongs to a class j and is classified as class i. In the above cost matrix of Table 2.1 the cost of classifying an example incorrectly should always be greater than the cost of classifying it correctly. Hence $c_{01} > c_{00}$ and $c_{10} > c_{11}$. Moreover, misclassification cost of a minority class example should always be greater than those of majority class examples. Elkan [22] gives a mathematical expression of misclassification cost.

$$L(x,i) = \sum_{j} P(j|x)C(i,j) \tag{2.1}$$

In Equation 2.1 $L(x,i)$ is the sum over the alternative possibilities of true class of x for each i. $C(i,j)$ is the misclassification cost when actual class is j and classifier predicts it in class i. $P(j|x)$ is the probability of classifying an instance x in class j. When $i = j$ the sample is correctly classified and misclassification cost is 0.

2.2.2 One-Class Learning

One-Class Learning also called as Recognition Based Approach is a method that takes only minority class into consideration [24]. It works by assuming that there are examples of only one class available in the whole dataset. The model for the classification of new examples are created only from minority class examples.

In one-class learning the idea is to measure the similarity between the examples of the minority class by applying some threshold value on the similarity value [25]. Raskutti et al. [24] shows that one-class learning is particularly useful when used on extremely unbalanced datasets composed of a high-dimensional noisy feature space. They argue that the one-class approach is related to aggressive feature selection methods, but is more practical since feature selection can often be too expensive to apply.

2.2.3 Active Learning on Border

Active learning is a method most regarded as the classification of unlabeled instances [26]. Active learning algorithm accesses very large amount of data in order to label them. However, because of selecting informative instances from a very small pool of data, it does not have to go through all the instances in the dataset. The informativeness of an instance can be measured as its distance from the decision boundary.

The instances which are farther away from the decision boundary bring no new information and can be classified very easily. However, the instances that are close to decision boundary are more informative. Ertekin et al. [26] suggested that the ratio

of imbalance near decision boundary is very small. The examples on the decision boundary can be classified by any machine learning classifier because ratio of majority and minority class examples is same.

2.2.4 Ensemble Learning

Ensemble Learning is a machine learning paradigm in which multiple models, such as classifiers and experts are combined to solve some classification problem. Unlike traditional machine learning classifiers which try to learn only *one* hypothesis from the training data, ensemble learning methods use a combination of different hypotheses to classify data. The success of an ensemble learner depends on the different classifiers that make up the ensemble learner.

If all the classifiers learn the same knowledge, then they all behave as one classifier and will bring no improvement in the classification. A good approach in building an ensemble classifier is first accurately build individual classifiers and then choose the best classifiers from them [27]. In the following sections we describe the most important methods used in ensemble learning.

Bagging

Bagging method was first introduced by Breiman [28] in 1994. The main idea in bagging method is to generate many versions of classifiers and then get an aggregated classifier from these classifiers. Using bootstrap sampling method different training datasets are created and the classifier is tested on these different training datasets in order to get new classifiers.

Bootstrap sampling is a method in which K new training datasets of size $n^{'}(n^{'} < n)$ are created from a TR of size n. If $n^{'} = n$ then the TR K_i has a chance of having 63.2% of newly generated samples to be unique and the rest will be replicated. If classifiers that are generated in bagging are considerably different from each other then bagging results in high accuracy.

Boosting

Boosting method is first introduced by Schapire [29] while answering hypothesis boosting problem (HBP). HBP was described by can be described Kearns [30] as

a question, if a set of weak learners create a strong learner. A strong learner can be described as [29] "given access to a source of examples of the unknown concept, the learner with high probability is able to output an hypothesis that is correct on all but an arbitrarily small fraction of the instances" and a weak learner can be described as [29] "a learner can produce an hypothesis that performs only slightly better than random guessing".

In boosting method weak learner is converted into a strong learner which achieves high accuracy in data classification. The main idea in boosting is that instead of learning a single weak classifier learn many weak classifiers that are good at different parts of the input space. The data classification can be done by weighted vote of each classifier about a given data sample. The weight distribution over classifiers about a data sample depends on their self-confidence levels of the classifier. Data classification can be done forcing learners to learn about the different parts of the input space. The final classifier can be obtained by a linear combination of the votes of different classifiers weighted by their strengths.

2.3 Classifier Evaluation Metrics

Evaluation metrics play a central role in classifier evaluation. Their purpose is to evaluate the goodness of a classifier in data classification. Like a traditional classifiers do not work well with imbalanced datasets, similarly traditional machine learning evaluation metrics, which work by considering the overall classification rate are not suitable for imbalanced datasets, as the minority class contributes very little to the overall data. A variety of evaluation metrics are developed based on the confusion matrix.

Table 2.2: Confusion Matrix.

	Actual Positive	**Actual Negative**
Predicted Positive	True Positive (TP)	False Positive (FP)
Predicted Negative	False Negative (FN)	True Negative (TN)

Some of the most commonly used evaluation metrics which are used for imbalanced datasets are Precision, F-measure, Geometric Mean, ROC Curve and AUC. Table 2.2 describes TP, FP, TN and FN which are used in below mentioned evaluation metrics.

Precision: It is the percentage of the positive (minority class) instances that are actually positive.

$$Precision = \frac{TP}{TP + FP}$$

Recall: Recall is based on understanding and a measure of relevance of data. It means that how much an algorithm return the relevant results, so it is actually the percentage of true positive patterns that are correctly classified by the classifier.

$$Recall = \frac{TP}{TP + FN}$$

F-measure: F-measure is the harmonic mean of precision and recall. A high F-measure value signifies high value of precision and recall [31].

$$F\text{-}measure = 2 \times \frac{precision \times recall}{precision + recall}.$$

Geometric mean: Geometric mean (G-mean) is a very well known evaluation metric for evaluating imbalanced dataset classifiers. It indicates the balance between majority and minority class instances in a dataset. It makes use of sensitivity (accuracy on positive examples) and specificity (accuracy on negative examples) [31].

$$Sensitivity = Recall$$

$$Specificity = 1 - \frac{FP}{TotalNegatives}$$

$$G\text{-}Mean = \sqrt{Sensitivity \times Specificity}$$

ROC and AUC: Receiver Operating Characteristics (ROC) and Area Under ROC (AUC) are the two very common metrics for the overall evaluation of a classifier.

ROC curve is a two-dimensional graph to select optimal model based on TP and FP rate. It represents changes in TP and FP rate as the decision boundary changes. The ROC curve shows that for any classifier the true positive rate cannot increase without increasing the FP rate [31]. The TP rate is the same as recall and false detection rate (FDR) is

$$FDR = \frac{FP}{TotalNegatives}$$

ROC gives visual indication of a classifier performance as each prediction result is represented as one point in ROC space. The area under the ROC curve (AUC) can be represented to summarize the performance of a classifier into a single metric.

3. PROPOSED APPROACH

In this chapter a new approach on data level to handle imbalanced data classification is proposed. This chapter in detail discusses how this new approach can be used and what are the advantages of this approach.

3.1 Overview of Proposed Approach

The main idea in this new approach is to generate new samples for the minority class distribution by making a random walk using some MCMC algorithm. Generating new samples by making a random walk in the minority class results in new examples which are similar to minority class samples yet they are different (not replicated) from already existing minority class examples. Although newly generated samples are different from already existing samples, there is a chance that some newly generated samples are replication of some other newly generated samples. In order to minimize this, many new samples are generated and some of them are selected randomly and added to the original training set.

In the original dataset each node has dependency on many other nodes. The dependency of nodes in the original dataset can be represented by a graph. We create a tree structure in order to keep minimum dependency of a node on other nodes. This tree is called as Chow-Liu (CL) Tree. The parent of a node is selected by calculating the mutual information (MI) of all the nodes between each other. Two nodes x_i and x_j will be in parent-child relationship if node x_j has the maximum MI value with node x_i. In such a scenario, node x_i will become the parent of node x_j. A sampling algorithm such as Gibbs sampling is applied on the CL tree to get new samples. After getting this new training data in which the minority class is oversampled, different machine learning classifiers are applied on the new training data to record their performance with new training data.

So our new approach is divided into following steps:

- Calculating Mutual information between minority class examples.
- Creating CL tree from the nodes of minority class.
- Applying some MCMC algorithm in order to generate new samples.
- Random selection of newly generated samples to avoid replication.
- Making a cross validation on new data sample.

3.2 Generating Chow-Liu Trees

This section describes how to calculate MIs between parameters of a network, which is used in generating CL tree given the training data. The CL tree will be later used in generating new samples for minority class.

3.2.1 Mutual Information and Chow-Liu Tree

CL tree was first introduced by Chow and Liu [32] while describing n-dimensional discrete probability distribution by a product of second-order distributions. Chow and Liu provided a very simple algorithm for creating an optimal tree. At each stage the algorithm calculates the MI between the nodes and adds a pair to the tree with maximum MI value.

MI is a measure of mutual dependence of the two random variables on each other. It measures how much two random variables share information with each other. The MI for two discrete random variables can be calculated by following relation.

$$I(X,Y) = \sum_{y \in Y} \sum_{x \in X} p(x,y) \log \frac{p(x,y)}{p(x)p(y)}, \tag{3.1}$$

where $p(x,y)$ is the joint probability of variable X and Y, and $p(x)$ and $p(y)$ are the marginal probabilities of variables X and Y respectively. The algorithm calculates

MI for each pair of random variables X and Y. As can be seen in the above relation the value of MI will always be non-negative. If two random variables do not share any information with each other then the value of MI between them will be 0. The value of MI shows the dependence of two random variables on each other. If the two random variables have high dependence on each other then the value of MI between them will be higher.

CL tree is created by making a minimal spanning tree (MST) by using Kruskal's algorithm [33]. The MST is created based on the MI values calculated earlier. The nodes which have the maximum value for the mutual information between each other are added as root of the tree and similarly the rest of the nodes are added as the children of already existing nodes in the tree.

3.2.2 Why Chow-Liu Tree is Important

CL tree describes a way to approximate joint probability distribution as a product of second-order conditional and marginal distribution. CL trees are extremely effective in making the complex relationships simple. A very simple CL tree is show in Figure 3.1.

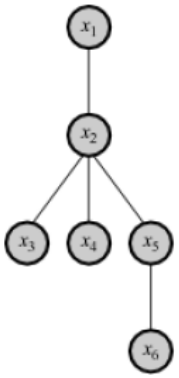

Figure 3.1: A Simple Chow-Liu Tree.

In Figure 3.1 each node of the tree has exactly one possible parent except the root node. The joint probability distribution for this tree can be very easily computed by

$$P(X_1, X_2, X_3, X_4, X_5, X_6) = P(X_6|X_5)P(X_5|X_2)P(X_4|X_2)P(X_3|X_2)$$
$$P(X_2|X_1)P(X_1).$$

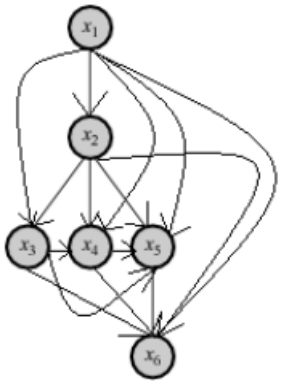

Figure 3.2: A Simple Bayesian Network.

Now consider a simple Bayesian network shown in Figure 3.2. The joint probability distribution in the above Bayesian network can be calculated by

$$P(X_1, X_2, X_3, X_4, X_5, X_6) = P(X_6|X_5X_4X_3X_2X_1)P(X_5|X_4X_3X_2X_1)$$
$$P(X_4|X_3X_2X_1)P(X_3|X_2X_1)(X_2|X_1)P(X_1).$$

Already it can be noticed that the number of parameters in calculating the joint probability distribution in Bayesian network is much more than in CL tree. The calculation for joint probability distribution becomes intractable as the number of parameters increases. An example of a very big Bayesian network can be seen in Figure 3.3.

Bayesian network in Figure 3.3 has almost 500 parameters and each of them have 4 values on average. So the total number of parameters will be 4^{500} in full joint probability distribution. In the real life systems we can have much more parameters than the network in Figure 3.3. Hence in order to make the calculation of joint probability distribution tractable in real life applications, it is very important to minimize the dependencies of nodes on each other.

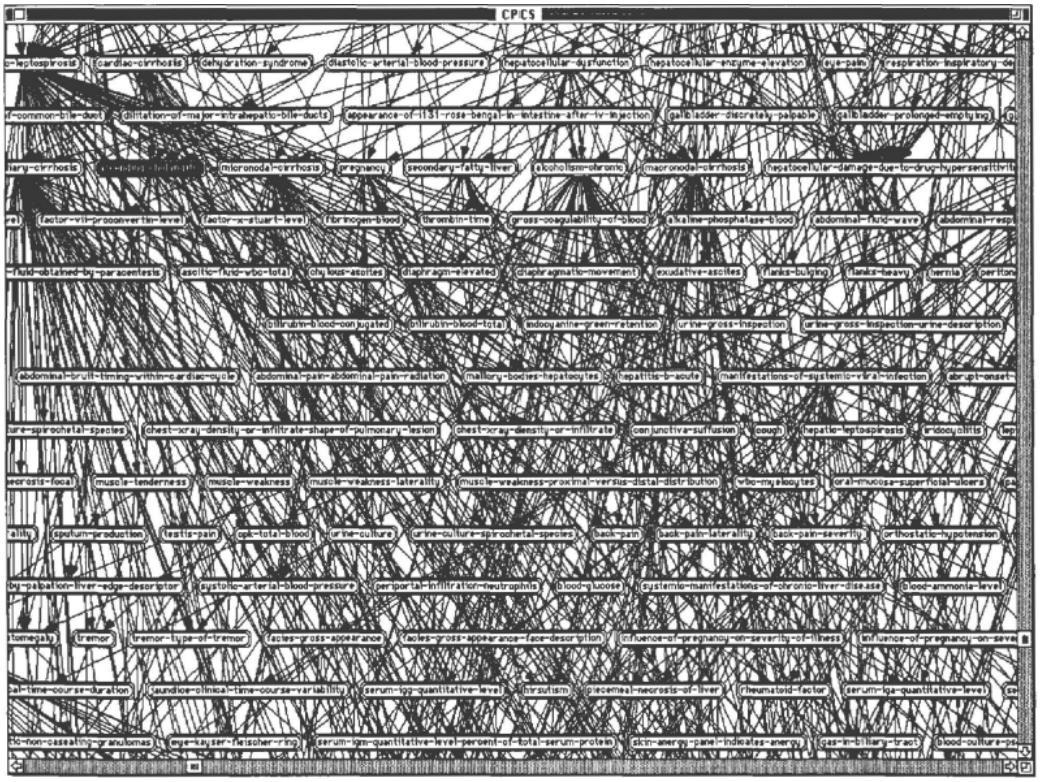

Figure 3.3: A Small Portion of the Computer-based Patient Case Simulation (CPCS) Belief Network in Netview Visualization.

3.2.3 Mutual Information through Empirical Probability

The goal of CL tree is to maximize the likelihood of data. Understanding the structure and dependence of parameters on each other in a network is very hard just by presenting the training data. Therefore, generating CL tree gives a good approximation of the structure and parameter dependence in a network. The main idea in creating CL tree for any given TR is to calculate MI and create a MST. MI can be calculated by (3.1), but calculating MI for parameters require joint probability distribution between the parameters. Instead of calculating the joint probability, it is much easier to find the empirical probability which can also be used instead of joint probability. Hence (3.1) is updated in a form which uses empirical probability distribution.

$$\hat{I}(X_i, Y_j) = \sum_{y \in Y} \sum_{x \in X} \hat{p}(x_i, y_j) \log \frac{\hat{p}(x_i, y_j)}{\hat{p}(x_i)\hat{p}(y_j)}, \tag{3.2}$$

where $\hat{p}(x, y)$ is the empirical joint probability distribution of variables x and y, similarly $\hat{p}(x)$ and $\hat{p}(y)$ are the empirical probability distributions of variables x and y. The empirical probability in (3.2) can be calculated by

$$\hat{p}(x_i, y_j) = \frac{Count(x_i, y_j)}{m}, \tag{3.3}$$

$$\hat{p}(x_i) = \frac{Count(x_i)}{m}, \tag{3.4}$$

$$\hat{p}(y_j) = \frac{Count(y_j)}{m}, \tag{3.5}$$

where $Count(x_i, y_j)$ is the number of occurrences of x_i with y_i together in minority class and m is the size of the minority class. Variables x_i and y_i represent different values that a parameter can take. For example, variable X represents a color variable then x_i represents some value that a color variable X can take like green. It should be noted, that the end goal is to maximize the data likelihood in minority class. Therefore, minority class samples are first extracted from the training data and then by (3.2) MI between parameters is calculated. As each parameter in the feature vector may have many values, hence (3.3) should be calculated for all the values of parameters X and Y. Equation 3.6 gives detailed formula to calculate MI between two parameters X and Y:

$$\hat{P}(X_1, X_2) = \sum_{x_i \in X_1} \sum_{x_j \in X_2} \hat{P}(X_1 = x_i, X_2 = x_j) \log \frac{\hat{P}(X_1 = x_i, X_2 = x_j)}{\hat{P}(X_1 = x_i)\hat{P}(X_2 = x_j)}. \tag{3.6}$$

3.3 Markov Chain Monte Carlo Algorithms

This section describes a family of sampling algorithm called as MCMC algorithms. The main idea in MCMC algorithms is to get samples from a probability distribution also called as target distribution by constructing a Markov Chain (MC) with stationary distribution described in Section 3.3.2. MCMC sampling idea was first described by Metropolis et al. [34], and later Hastings [35] presented a generalization of the algorithm.

Consider we have a target distribution $\pi(x)$, which is known only up to a certain multiplicative constant. If the target distribution is very complex to sample from

it directly, then an indirect method is used to sample the distribution π which uses an irreducible and aperiodic MC which has the stationary distribution $\pi(x)$ [36]. In order to sample from this distribution we have to run the chain sufficiently long so that it achieves its stationary distribution.

3.3.1 Markov Chains

A MC (can be considered as a state machine) is mathematical system which consists of a finite number of states. It undergoes transitions from one state to others states. The edges between different states usually represent the probability of making a transition to a certain state. Making a transition of MC is a random process, in which moving to next state is only dependent on the current state. Therefore, MCs are considered as memoryless systems. MC defines a probabilistic transition model $T(x \to x')$ over all states x characterized by the property:

$$\sum_{x'} T(x \to x') = 1. \tag{3.7}$$

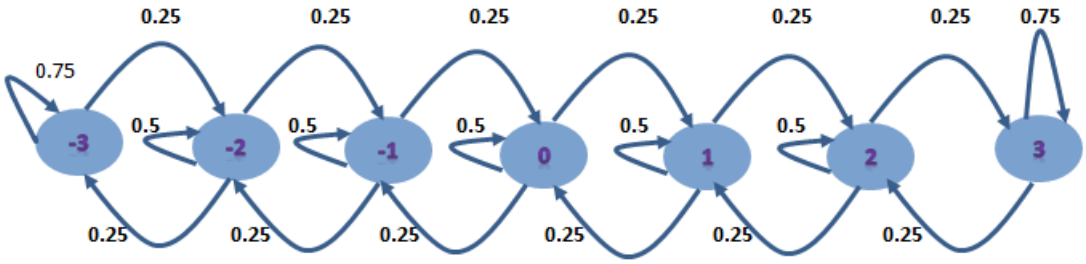

Figure 3.4: A Simple Markov Chain.

Lets consider a hypothetical agent at state 0 at time $t = 0$. Figure 3.4 shows a simple MC for an agent with transition probabilities for all the states. It can be verified from Figure 3.4 that the sum of the transition probabilities for every state at each time stamp sums up to 1. The probability for the agent to be in the state x' at any time $t + 1$ can be given by

$$P^{(t+1)}(X^{(t+1)} = x') = \sum_{x} P^{(t)}(X^{(t)} = x)T(x \rightarrow x'). \tag{3.8}$$

Equation 3.8 defines a recurrence relation which calculates the probability for agent to be in state x' at any time $t+1$. Let x be a state through which agent can reach to the state x' with a single transition. Then the total probability of being in state x' at time $t+1$ is *the sum of the products of each probability of being in state x at time t and the corresponding transition probability of moving to state x' from the state x, over all the states x.* The total probability for the agent starting from the state 0 to all the possible states at first three time steps is given by

Table 3.1: Probability of Being on different states for First Three Time Steps.

	-2	-1	0	1	2
$P^{(0)}$	0	0	1	0	0
$P^{(1)}$	0	0.25	0.5	.25	0
$P^{(2)}$	$0.25^2 = 0.0625$	$2 \times (0.5 \times 0.25) = 0.25$	$0.5^2 + 2 \times 0.25^2 = 0.375$	$2 \times (0.5 \times 0.25) = 0.25$	$0.25^2 = 0.0625$

3.3.2 Properties of Markov Chains

Sampling through MCMC algorithm that uses MCs, should satisfy some important properties of MC. In order to generate samples which can be used as new samples of minority class, MC makes a random walk sufficiently long such that the chain reaches a state of stationary distribution.

Stationary Distribution

A MC is said to reach its stationary distribution when the following relation holds.

$$\pi(x') \approx \pi(x) \tag{3.9}$$

$$\pi(x') = P^{(t+1)}(x') = \sum_{x} \pi(x) T(x \rightarrow x') \tag{3.10}$$

MC reaches its stationary distribution when Equation 3.9 holds, which states that the probability of being in state x' at time $t+1$ is approximately equal to probability of being in state x' at time t. It is noticed that not all the MCs reach the stationary distribution, a MC reaches the stationary distribution when it is regular.

Regular Markov Chain

A MC is said to be regular if there exists k such that for every pair of states x and x', the probability of reaching to x' from state x is greater than 0 in exactly k steps. When a markov chain is regular it is guaranteed to converge to a unique stationary distribution provided that the chain runs sufficiently long. A MC can be regular when the following two sufficient conditions hold.

- Every two states are connected.
- For every state, there exits a self transition.

Once the chain reaches a unique stationary distribution, samples are collected for the minority class. After collecting one sample next few samples should be thrown away because once the chain reaches stationary distribution, the adjacent samples are highly correlated with each other, so by collecting the samples which are further away from each other, replication in the samples can be avoided.

Once CL tree is generated as described in Section 3.2, the next step is generate samples using some MCMC algorithm. In our approach we are using Gibbs sampling to generate samples.

3.3.3 Gibbs Sampling

Gibbs sampling is a MCMC algorithm which is used to generate samples from a distribution for which the direct sampling is not possible. In our approach we have CL tree which acts as a MC on which gibbs sampling is applied. The MI values that are calculated earlier between the parameters act as the transition probabilities

for moving from one state to the next state in the CL tree. In order, to collect the samples using Gibbs sampling algorithm, we start by randomly selecting a node from the CL tree. The successive states are randomly selected according to the transition probabilities of the nodes. Similarly the current state of the selected successive node given by the current state of the current node is also selected by the transition probabilities of the states of the selected successive state (each state consist of different values, these values are described as the states of the state here). This process is repeated for all the nodes in the CL trees. Once the process has been repeated for all the nodes, the current values of all the nodes in the tree give us one sample of the minority class.

Algorithm 1 describes how to draw the successive node and the value for that node. In order to get one sample of minority class, we run Algorithm 1 for all the nodes in the CL tree.

Algorithm 1 GenerateSample(root). Generate one sample by making a random walk starting from the random node.

```
1: current = root
2: sum_trans = 0
3: sum_values = 0
4: trans_randNo = 0
5: value_randNo = 0
6: iteration_count = 0
7: total_iterations = 100000000
8: while iteration_count ≤ total_iterations do
9:    for i = 1 → neighbor.len do
10:      sum_trans = sum_trans + GetTrans(current, neighbor[i])
11:   end for
12:   transition_randNo = RANDOM(0, sum_trans)
13:   for p = 1 → neighbor.len do
14:      trans_randNo = trans_randNo − GetTrans(current, neighbor[p])
15:      if trans_randNo ≤ 0 then
16:         break
17:      end if
18:   end for
19:   next ← neighbor[p]
20:   values ← next.values
21:   for i = j → values.len do
22:      sum_values = sum_values + GetValue(next, values[j])
23:   end for
24:   value_randNo = RANDOM(0, sum_values)
25:   for k = 1 → values.len do
26:      value_randNo = value_randNo − GetValue(next, values[j])
27:      if value_randNo ≤ 0 then
28:         break
29:      end if
30:   end for
31:   current = next
32:   current.currentValue = values[k]
33:   iteration_count = iteration_count + 1
34: end while
```

Algorithm 1 generates samples for the minority class. The **total_iterations** here is described as a big number which shows that the algorithm should execute sufficiently many loop iterations for chain to achieve the stationary distribution. The **total_iterations** number can be different for different MCs. Variable **neighbor** is an array of nodes which contains those nodes on which current node may jump to in the next time stamp. Similarly variable **values** is an array of different values from which one of the value is assigned to the node as the current value. The function **GetTrans** return the transition probability from the current node to one of the neighboring nodes. Similarly the function **GetValue** return the probability of being in a certain state once the next node is selected. Once the new value is assigned to the next node, all the process is repeated from the newly assigned current node.

While generating CL tree some nodes are totally independent of other nodes (their MI value will be zero). These nodes are not added in our CL tree and ultimately will not be in our MC. Hence there will be no way to include these nodes in the sampling process while generating samples. However, a sample should always have all the node values in it. Hence we randomly draw some value for these nodes and add these values in the samples. We collect a lot of samples and then select randomly from them in order to avoid the correlated samples.

4. EMPIRICAL EVALUATION

This chapter discusses the experiments performed using the approach described in the previous chapter. First the data used in the experiments and its characteristics are given. We describe how the minority class from a given TR is identified. Next we describe the experimental setup and any third party tools and existing machine learning classifiers, which helped us perform the experiments. Then the empirical comparison between our approach and some already built approaches is given. Finally we analyze the results of our approach.

4.1 Data Characteristics

In this section we describe the data and its characteristics that we have used in our experiments. Both the datasets used in the experiments are collected from machine learning repository of University of California Irvine (UCI). The two datasets used from UCI repository in the experiments are Nursery and Car datasets. The main reason for using these two datasets is that the minority class is very easily identified in these two datasets just by looking at the datasets.

Table 4.1: Nursery Dataset.

Class	**Instances**	**Percentage**
not_recom	4320	33.333%
recommend	2	0.015%
very_recom	328	2.531%
priority	4266	32.917%
spec_prior	4044	31.204%

Nursery dataset is described in Table 4.1. It consists of 12960 data samples and it is divided into five classes with percentages of each class as described in Table 4.1. By analyzing the dataset we immediately see that there are two minority classes, which are **recommend** and **very_recom**. The recommend class has only 2 instance in the whole dataset. As we are generating new samples for minority class based on

the current examples in the minority, so we do not have much information here to generate new samples from these two samples. Hence we ignore recommend class and consider very_recom class as our minority class. The classes tell how likely it is for a child to get admission in the nursery school for example, very_recom class shows that it is highly recommended for a child to get admission in nursery school.

Nursery database was derived from a hierarchical decision model originally developed to rank applications for nursery schools. It was used during several years in 1980's when there was excessive enrollment to these schools in Ljubljana, Slovenia, the rejected applications frequently needed an objective explanation. The final decision depended on three parameters: occupation of parents and child's nursery, family structure and financial standing, and social and health picture of the family.

Table 4.2 below shows the class distribution of the car dataset.

Table 4.2: Car Dataset.

Class	**Instances**	**Percentage**
unacc	1210	70.023%
acc	384	22.222%
good	69	3.995%
vgood	65	3.760%

Car dataset describes the overall evaluation criteria for a car. The overall condition of the car is divided on two main features. Car overall price and the technical characteristics of the car. The price is further divided into buying and maintenance cost, similarly technical characteristics are divided into safety, number of doors, person capacity and the size of the luggage boot. So the overall condition of the car is divided on six features on the base level. The final condition values are unacceptable, acceptable, good, and very-good.

There are two minority classes in this dataset: good and very-good. However, as our approach deals with only one minority class at a time, hence very-good class is considered as the minority class in this dataset.

4.2 Experimental Setup

This section describes the experimental setup for the experiments, file formats used and the traditional machine learning classifiers to classify the oversampled dataset.

This thesis also uses a third party software which has many built in machine learning classifiers.

Once the oversampled minority class is generated by the our approach, it is shuffled with the original dataset to create new dataset, which is used for classification. We have used a third party software called WEKA [37]. WEKA (*Waikato Environment for Knowledge Analysis*) is a free and popular suite of machine learning software written in Java and developed by University of Waikato, New Zealand. WEKA uses a special file format called *.arff* as its input file. The newly generated dataset is automatically converted into *arff* format, which can be used by WEKA.

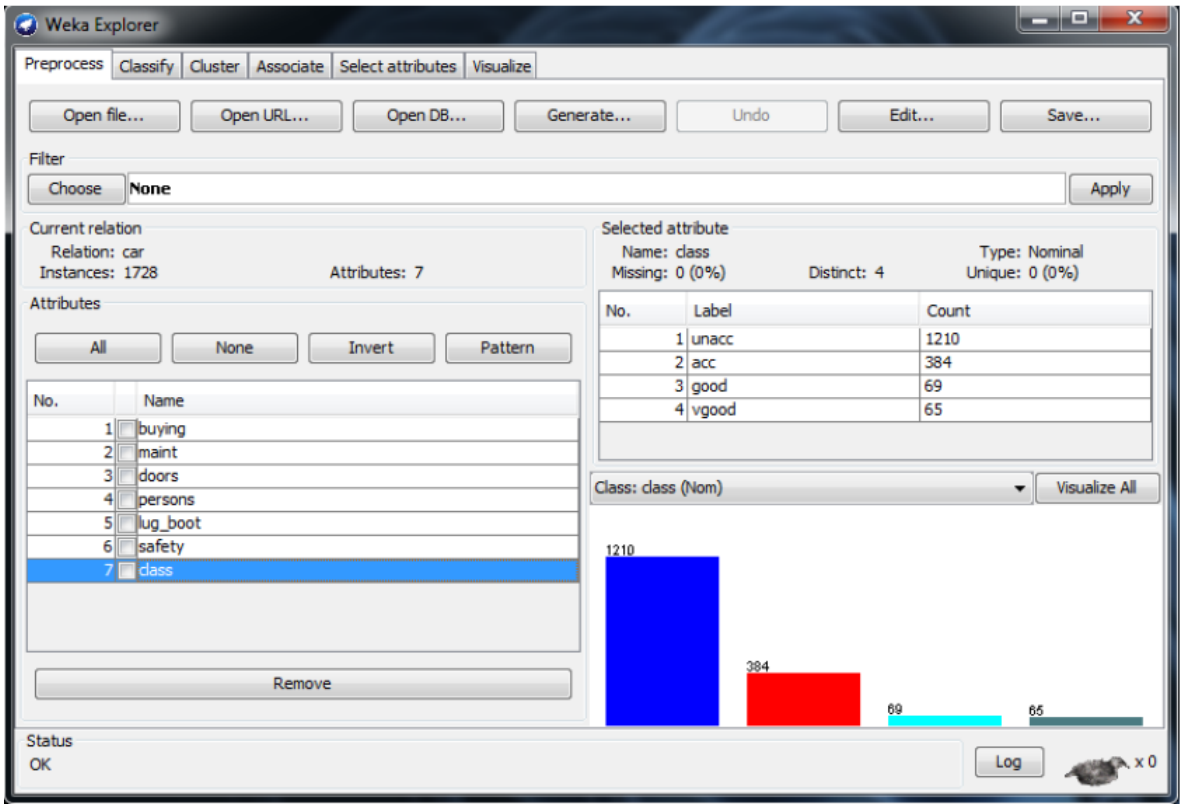

Figure 4.1: WEKA Car Dataset.

Figure 4.1 shows the car dataset loaded in WEKA. In the left panel WEKA shows all the parameters that affect the final decision. On the right side it shows the class distribution for different values. WEKA contains algorithms for both supervised and unsupervised classification task. Filters can also be applied on a dataset which includes shuffling, normalization and some oversampling techniques.

Once the data is loaded in WEKA, the next step is to apply some supervised machine learning classifier on the data and analyze the results. How data oversampling

helped in correctly classify the minority class? In our approach oversampled data is classified by J48, PART and SMO algorithms, these algorithms already exist in WEKA.

J48 is a decision tree learner. It is an open source Java implementation of C4.5 algorithm in WEKA. C4.5 is an algorithm used to generate a decision tree developed by Ross Quinlan [38]. C4.5 is an extension of Quinlan's earlier ID3 algorithm. The decision trees generated by C4.5 can be used for classification, and for this reason, C4.5 is often referred to as a statistical classifier. Similar to J48, PART is a decision rule based learner in WEKA. SMO is the implementation of support vector machine (SVM) [39] in WEKA. SVM is a supervised learning model that analyzes data and recognizes patterns, used for classification and regression analysis.

In order to classify the dataset we cross validated the whole dataset in ten folds. It means that the whole dataset is divided randomly in ten parts and nine of them together are used as TR and one is used as a test set at a time.

4.3 Results Format

WEKA outputs the results into a matrix form called Confusion Matrix. A confusion matrix is an $n \times n$ matrix, where n is the number of different classes. In the matrix each element n_{ij} represent, the element which is classified in class ω_j when its true class is ω_i.

$$C = \begin{bmatrix} 50 & 0 & 0 \\ 0 & 47 & 3 \\ 0 & 1 & 49 \end{bmatrix}$$

An example of the confusion matrix is shown above. There are total three classes to classify and each class have 50 examples, and in total there are 150 examples. It can be seen from the confusion matrix that four examples are misclassified. Based on the above confusion matrix we can estimate the classification error by the following relation:

$$\hat{E}(\alpha) = \frac{\sum_i^c \sum_j^c n_{ij} - \sum_i^c n_{ii}}{\sum_i^c \sum_j^c n_{ij}}, \tag{4.1}$$

where $\sum_i^c n_{ii}$ are those examples which are correctly classified, hence they should be subtracted from the remaining data samples. By analyzing the confusion matrix we can estimate that how well the oversample technique has performed. The test error in the above example is calculated by Eq. 4.1,

$$\hat{E}(\alpha) = \frac{150 - 146}{150} = 0.0266.$$

4.4 Empirical Results

This section describes the results of our approach on nursery and car evaluation datasets. The results are described by confusion matrix described in Section 4.3.

4.4.1 Nursery Dataset Results

Nursery dataset has 12960 data samples and 328 of them are minority class samples. Minority class is oversampled by ≈600%. The minority class size in the oversampled dataset will become 1947 and the total size of the dataset becomes 14579. Below are the results shown in the confusion matrix form for the classifiers J48, PART, and SMO one at a time.

Confusion matrices below show the results of applying J48 on the original and oversampled nursery dataset.

$$\begin{bmatrix} 4320 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 238 & 90 & 0 \\ 0 & 0 & 56 & 4049 & 161 \\ 0 & 0 & 0 & 73 & 3971 \end{bmatrix} \begin{bmatrix} 4320 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 1830 & 117 & 0 \\ 0 & 0 & 185 & 3936 & 145 \\ 0 & 0 & 0 & 76 & 3968 \end{bmatrix}$$

The minority class classification results are highlighted in the above confusion matrices. In the original dataset 238 minority class samples (MCS) are correctly classified out of 328 examples and 90 samples are incorrectly classified. However, in the oversampled dataset 1830 MCS are correctly classified out of 1947 and 117 examples are incorrectly classified. The accuracy for the minority classification with J48 for the

original dataset is 72.560% and for the oversampled dataset is 93.990%.

The matrices below show the results of applying PART classifier on nursery dataset.

$$\begin{bmatrix} 4320 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 322 & 6 & 0 \\ 0 & 0 & 16 & 4228 & 22 \\ 0 & 0 & 0 & 57 & 3987 \end{bmatrix} \begin{bmatrix} 4320 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 1806 & 141 & 0 \\ 0 & 0 & 130 & 4104 & 32 \\ 0 & 0 & 1 & 65 & 3978 \end{bmatrix}$$

PART classifier with the original dataset classifies 322 MCS correctly out of 328 and 6 MCS are classified incorrectly. However, 1806 MCS are correctly classified out of 1947 and 141 MCS are misclassified. The accuracy with PART for original dataset is 98.172% and for oversampled dataset is 92.758%.

The results for SMO classifier are shown below in the confusion matrices.

$$\begin{bmatrix} 4320 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 236 & 92 & 0 \\ 0 & 0 & 52 & 3801 & 413 \\ 0 & 0 & 0 & 338 & 3706 \end{bmatrix} \begin{bmatrix} 4320 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 1835 & 112 & 0 \\ 0 & 0 & 234 & 3623 & 409 \\ 0 & 0 & 0 & 346 & 3698 \end{bmatrix}$$

SMO classifies 236 MCS correctly with the original dataset and 92 MCS are misclassified. However, with the oversampled dataset SMO classifies 1835 MCS correctly and misclassifies 112 examples. The overall accuracy of the minority class classification in original dataset is 71.951% and with the oversampled dataset the classification accuracy is 94.247%.

WEKA also gives the classifier evaluation measures described in Section 2.3 like TP rate, FN rate, F-measure for a specific dataset. J48 evaluation measures for the oversampled nursery dataset are given in Table 4.3 .

The same results are obtained for the original dataset for the nursery data, apart for the minority class the true positive rate for the **priority** class is 0.94 and for **spec_prior** has 0.98 with J48. The classification rate for the majority class with oversampled dataset remains same for most of the majority classes. However, classification rate with oversampled data is increased significantly.

Table 4.3: J48 Nursery Dataset Evaluation Measures.

Class	TP Rate	FP Rate	Precision	Recall	F-Measure
not_recom	1	0	1	1	1
recommend	0	0	0	0	0
very_recom	0.94	0.015	0.907	0.94	0.923
priority	0.923	0.019	0.953	0.923	0.938
spec_prior	0.981	0.014	0.965	0.981	0.973

4.4.2 Car Evaluation Dataset Results

Car Evaluation dataset consists of 1728 data samples and the minority class has 65 examples. As there are two minority classes in this dataset, our approach considers only one minority class and treat all the other data samples as majority class instances. The algorithm oversamples the minority class until the number of minority class samples are close to some other class in the dataset. Minority class is oversampled $\approx 450\%$ and 287 new minority class instances are added to the dataset. The oversampled dataset has 2015 examples and 352 of them are minority class examples. The results of applying classifiers on this dataset is described in confusion matrices below:

$$\begin{bmatrix} 1164 & 43 & 3 & 0 \\ 33 & 333 & 11 & 7 \\ 0 & 17 & 42 & 10 \\ 0 & 3 & 5 & 57 \end{bmatrix} \begin{bmatrix} 1125 & 46 & 3 & 36 \\ 30 & 336 & 11 & 7 \\ 0 & 17 & 42 & 10 \\ 5 & 4 & 6 & 337 \end{bmatrix}$$

The confusion matrices above show the result of applying J48 classifier on original and oversampled datasets. In the original dataset 57 MCS are correctly classified and 8 MCS are misclassified. However, in the oversampled dataset 337 MCS are correctly classified out of 352 MCS and 15 are misclassified. The resultant classification accuracy for the original dataset is 87.692% and for oversampled dataset is 95.738%.

Confusion Matrices below show results of applying PART classifier on original and oversampled car dataset.

$$\begin{bmatrix} 1180 & 26 & 4 & 0 \\ 6 & 360 & 16 & 2 \\ 0 & 15 & 51 & 3 \\ 0 & 1 & 0 & 64 \end{bmatrix} \begin{bmatrix} 1149 & 33 & 2 & 26 \\ 4 & 360 & 13 & 7 \\ 1 & 11 & 54 & 3 \\ 6 & 3 & 4 & 339 \end{bmatrix}$$

It can be deduced from the confusion matrices above that, with PART classifier all the MCS are correctly classified except 1 MCS. The classification accuracy for the original data set is 98.461% and, with the oversampled dataset PART classifier performs slightly better than J48 classifier. In the oversampled dataset 339 MCS are correctly classified with the classification accuracy of 96.306% and 13 MCS are misclassified.

Confusion matrices below show the results of applying SMO classifier on the car data set.

$$\begin{bmatrix} 1161 & 48 & 1 & 0 \\ 31 & 341 & 10 & 2 \\ 0 & 11 & 55 & 3 \\ 0 & 0 & 2 & 63 \end{bmatrix} \begin{bmatrix} 1123 & 42 & 2 & 43 \\ 32 & 284 & 11 & 57 \\ 0 & 14 & 51 & 4 \\ 27 & 28 & 5 & 292 \end{bmatrix}$$

The classification results for SMO conclude that with oversampled dataset SMO results in poor classification and with the classification accuracy of 82.954%. The classification accuracy for the original dataset is 96.923%, only 2 examples are misclassified in the original dataset. In the oversampled dataset 292 examples are correctly classified and 60 examples are misclassified.

Table 4.4: J48 Car Evaluation Dataset Evaluation Measures.

Class	**TP Rate**	**FP Rate**	**Precision**	**Recall**	**F-Measure**
unacc	0.93	0.043	0.97	0.93	0.949
acc	0.875	0.041	0.834	0.875	0.854
good	0.609	0.01	0.677	0.609	0.641
vgood	0.957	0.032	0.864	0.957	0.908

Table 4.4 shows the classifier J48 evaluation measures for the car evaluation dataset. The same evaluating metrics are also obtained for the original dataset. True positive rate with J48 for oversampled dataset is 0.957 and for original dataset is 0.877. For

the other majority class true positive rate for the oversampled data is given in Table 4.4. True positive rate in the original dataset for class **unacc** is 0.962, for **acc** is 0.867, and for **good** is 0.609. It can also be deduced from the confusion matrices above that PART classifier always results in better classification accuracy with the original dataset.

4.5 Comparison with Other Approaches

This section makes a comparison of our new approach with SMOTE oversampling approach described in Section 2.1.1. Detailed results of the comparison are given in Tables 4.5 and 4.6.

We have used SMOTE to oversample nursery and car evaluation datasets. The number of newly generated samples for minority class with SMOTE is the same as the number of examples that we have generated for our approach. Table 4.5 shows the classification results for oversampled nursery dataset with SMOTE oversampling technique.

Table 4.5: SMOTE Oversampled Nursery Dataset Classification.

	Correctly Classified	**Misclassified**	**Accuracy**
J48	1897	48	97.532%
PART	1940	5	99.742%
SMO	1883	62	96.812%

Table 4.6 shows the results for the oversampled car evaluation dataset with SMOTE. It can be deduced from the Table 4.6 that PART and SMO result in 100% classification accuracy and J48 misclassifies 2 MCSs.

Table 4.6: SMOTE Oversampled Car Evaluation Dataset Classification.

	Correctly Classified	**Misclassified**	**Accuracy**
J48	351	2	99.433%
PART	353	0	100%
SMO	353	0	100%

It can be deduced from the Tables 4.5 and 4.6 that the results of the SMOTE over-sampling technique are slightly better than our approach in both the datasets with all three classifiers. The samples generated by our approach is largely dependent on the structure of CL tree. In the next chapter we describe some enhancement methods which can further improve the results of our approach.

5. DISCUSSION AND FURTHER WORK

In this chapter we will discuss the effect of applying our oversampling approach on the minority class classification. This chapter also presents two improvement techniques which can be used to improve the classification results.

5.1 Structure of Chow Liu tree

The newly generated samples are largely dependent on the structure of CL tree. Therefore, the structure of the CL tree ultimately have a big impact on the the classification of the minority class examples. The difference between the newly generated minority class examples is directly related to the structure of the CL tree. If the tree is very thin and tall then it may take many iterations to sample all the nodes in the CL tree as compared to the a thick and shallow tree.

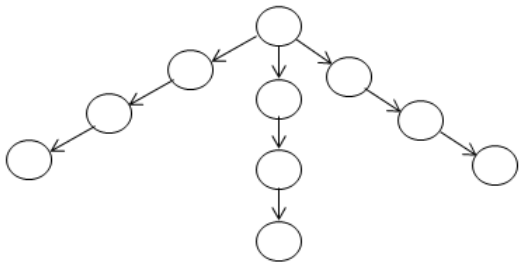

Figure 5.1: Tall Chow-liu Tree.

Consider the CL tree in Figure 5.1. Each node has only one child except the root node. New value of a node is drawn by making a random jump from one node to its adjacent nodes based on the empirical probabilities distribution. If the algorithm is generating new value for a leaf node which is on one end of the CL tree, then it may take very long to generate new value for the leaf node which is on the other end of the tree. The probability of generating new samples in this way for every node is very low in a short period of time. Therefore, it takes many iterations of the algorithm to generate new samples for every node. The result of this type of tree structure is

that for most of the time it generates new values for some nodes and does not generate any new value for some nodes in the tree. It results in two main disadvantages:

- For many iterations we get replicated samples.
- After running the chain for a long time, in the end we have to randomly select samples from all the newly generated samples.

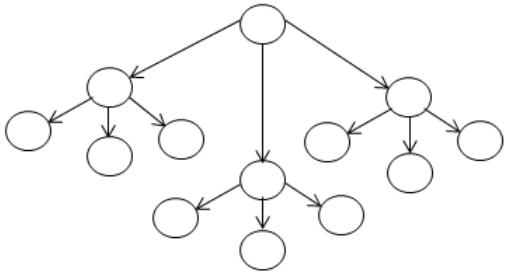

Figure 5.2: Shallow Chow-liu Tree.

Now consider the CL tree in the Figure 5.2. This tree is much more thick and shallow as compared to the tree in Figure 5.1. If the resultant CL tree is thick and shallow, then the probability of generating new values for nodes is much more in a short period of time and ultimately needs few iterations. However, the structure of CL tree represents the dependence of nodes on each other. Hence, we should alter the tree structure in such a way that the altering will not affect the dependence relation in nodes.

Currently our approach uses a very naive method to generate the CL tree and in most cases is similar to that in Figure 5.1. However, more heuristics can be applied to make the tree without affecting the nodes dependence on each other.

5.2 Mutation with Majority Class

In our current sampling approach, new values for a node are only drawn from the existing minority class. In following this approach newly generated samples are always those sample which already exist in the minority class but with different combination of values for each node. For example, if the total number of parameters in a

dataset is ten and on average each parameter has five different values then the total number of different samples are 5^{10}. If in the minority class each parameter on average contribute with its two values, then new values for a parameter is drawn only from those two values that exist in the minority class for this specific parameter. The total number of different newly generated examples are only 2^{10}, after that all the newly generated samples are replication of the existing samples.

We can always mutate minority class samples with some new value for the parameters in minority class samples. These new values should be some value outside the current minority class distribution. The addition of new value in the minority class results in generating many new minority class samples, which does not already exit in the minority class. Adding these new samples gives the minority class some variety, so if there are some minority class samples that are added in the original dataset, then this approach may also cover those minority class samples.

6. CONCLUSION

In this chapter we make some final comments about this new oversampling technique for the classification of the minority class. We will make a very brief final comparison of original dataset and oversampled dataset classification results.

In this thesis we have introduced a new approach which involves oversampling of minority class by making a random walk in order to generate new samples for the minority class. We have then compared our approach with original dataset classification and with SMOTE oversampling technique. The comparison results described in Sections 4.4 and 4.5 reveal that our oversampling approach is generally better than original dataset and not better than SMOTE oversampling technique.

Overall the classification accuracies of our approach with both the datasets are very good with overall accuracy of more than 92% with most of the classes. Our oversampling technique can be further improved by making more enhancements described in Sections 5.1 and 5.2.

BIBLIOGRAPHY

[1] S. Wang, "A Comprehensive Survey Of Data Mining Based Fraud Detection Research," *International Conference on Intelligent Computation Technology and Automation (ICICTA), vol. 1, pp. 50–53, May 2010.*

[2] C. M.D. and J. Serrano, "A Multistrategy Approach For Digital Text Categorization From Imbalanced Documents," *ACM SIGKDD Explorations Newsletter - Special issue on learning from imbalanced datasets, vol. 6, pp. 70-79,Issue 1, June 2004.*

[3] M. Kubat, R. Holte, and S. Matwin, "Machine Learning For The Detection of Oil Spills in Satellite Radar Images." *Journal Machine Learning - Special issue on applications of machine learning and the knowledge discovery process archive, vol 30, pp. 195–215, March, 1998.*

[4] R. Barandela, J. Sanchez, V. Garcia, and E. Rangel, "Strategies For Learning In Class Imbalance Problems," *The Journal Of Pattern Recognition Society, vol. 36, pp. 849–851, August, 2002.*

[5] X. Guo, Y. Yin, C. Dong, G. Yang, and G. Zhou, "On the Class Imbalance Problem," *Fourth International Conference on Natural Computation, pp. 192–201, October 2008.*

[6] N. Japkowics, "The Class Imbalance Problem: Significance and Strategies." *In Proceedings of the 2000 International Conference on Artificial Intelligence (ICAI), pp. 111–117, 2000.*

[7] S. Wang and X. Yao, "Multi-Class Imbalance Problems: Analysis and Potential Solutions," *IEEE TRANSACTIONS ON SYSTEMS, MAN AND CYBERNETICS, PART B, vol. 42, pp. 1119–1130, August 2012.*

[8] S. Wang, H. Chen, and X. Yao, "Negative correlation learning for classification ensembles," *International Joint Conference on Neural Networks, WCCI. IEEE Press, pp. 2893–2900, 2010.*

[9] V. Garcia, J. Sanchez, R. Mollineda, R. Alejo, and J. Sotoca, "The Class Imbalance Problem In Pattern Classification And Learning. 2007."

[10] A. Estabrooks, T. Jo, and N. Japkowicz, "A Multiple Resampling Methods for Learning From Imbalanced Data Sets." *Computational Intelligence,vol. 20, Issue 1, pp. 18–36, February 2004.*

[11] N. Chawla, K. Bowyer, L. Hall, and W. Kegelmeyer, "SMOTE: Synthetic Minority Over-sampling Technique." *Journal of Artificial Intelligence Research vol. 16, pp. 321-357, 2002.*

[12] T. Cover and P. Hart, "Nearest-Neighbor Pattern Classification." *IEEE Transactions on Information Theory, vol. 13, pp. 21–27, January 1967.*

[13] H. Han, W. Wang, and B. Mao, "Borderline-SMOTE: A New Over-Sampling Method in Imbalanced Data Sets Learning." *Advances in Intelligent Computing , pp. 878–887, 2005.*

[14] C. Bunkhumpornpat, K. Sinapiromsaran, and C. Lursinsap, "Safe-Level-SMOTE: Safe-Level-Synthetic Minority Over-sampling TEchnique for Handling the Class Imbalanced Problem." *Proceedings of the 13th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining pp. 475-482, 2009.*

[15] S. Barua, M. Islam, X. Yao, and K. Murase, "MWMOTE - Majority Weighted Minority Oversampling Technique for Imbalanced Data Set Learning." *IEEE Transactions on Knowledge and Data Engineering, 26 Nov. 2012.*

[16] J. Laurikkala, "Improving Identification of Difficult Small Classes By Balancing Class Distribution." *AIME '01 Proceedings of the 8th Conference on AI in Medicine in Europe: Artificial Intelligence Medicine, pp. 6–66, 2001.*

[17] I. Tomek, "Two Modifications of CNN." *IEEE Transactions on Systems, Man, and Cybernetics - TSMC , vol. 6, pp. 769–772, 1976.*

[18] S. Garcia, J. Cano, A. Fernandez, and F. Herrera, "A proposal of evolutionary prototype selection for class imbalance problems." *7th International Conference on Intelligent Data Engineering and Automated Learning, pp. 1415–1423, 2006.*

[19] P. Hart, "Nearest-Neighbor Pattern Classification." *IEEE Transactions on Information Theory, vol. 14, pp. 515–516, May 1968.*

[20] S. Yen and Y. Lee, "Cluster-based Under-sampling approaches for Imbalanced data Distribution." *Expert Systems with Applications: An International Journal archive, vol 36, pp. 5718–5727, Issue 3, April, 2009.*

[21] H. He and E. Garcia, "Learning from Imbalanced Data." *IEEE Transactions on Knowledge and Data Engineering, vol. 21, pp. 1263–1284, September 2009.*

[22] C. Elkan, "Foundation of Cost Sensitive Learning." *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, vol. 2, pp. 973–978, 2001.*

[23] C. Ling and V. Sheng, "Supervised Versus Unsupervised Binary-Learning by Feed-forward Neural Networks." *Encyclopedia of Machine Learning. C. Sammut (Ed.). Springer, 2011.*

[24] B. Raskutti and A. Kowalczyk, "Extreme Re-balancing for SVMs: a case study." *ACM SIGKDD Explorations Newsletter - Special issue on learning from imbalanced datasets Homepage archive, vol. 6, pp. 60–69, 1, June 2004.*

[25] C. Ling and V. Sheng, "Cost-Sensitive Learning and the Class Imbalance Problem." *Encyclopedia of Machine Learning. Springer. 2008.*

[26] S. Ertekin, J. Huang, L. Bottou, and C. Giles, "Learning on the border: Active learning in imbalanced data classification," *Proceedings of the sixteenth ACM Conference on Information and Knowledge Management, pp. 127–136, 2007.*

[27] Z. Ding, "Diversified Ensemble Classifiers for Highly Imbalanced Data Learning and their Application in Bioinformatics," *Computer Science Department, Georgia State University, 2011.*

[28] L. Breiman, "Bagging Predictors," *Machine Learning, pp. 132–140, 1996.*

[29] R. Schapire, "The Strength of Weak Learnability." *Machine Learning (Boston, MA: Kluwer Academic Publishers) 5 (2): pp. 197–227, 1990.*

[30] M. Kearns, "Thoughts on Hypothesis Boosting." *Unpublished manuscript (Machine Learning class project), December 1988.*

[31] S. Phung, A. Bouzerdoum, and G. Nguyen, "Learning pattern classification tasks with imbalanced data sets." *In P. Yin (Eds.), Pattern recognition. Vukovar, Croatia: In-Teh, pp. 193–208, 2009.*

[32] C. Chow and C. Liu, "Approximating Discrete Probability Distributions with Dependence Trees." *IEEE Transactions on Information Theory IT-14 (3), pp. 462-467, May 1968.*

[33] J. Kruskal, "On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem." *Proceedings of the American Mathematical Society, Vol 7, No. 1 , pp. 48–50, Feb, 1956.*

[34] N. Metropolis, A. Rosenbluth1, M. Rosenbluth, A. Teller, and E. Teller, "Learning from Imbalanced Data." *The Journal of Chemical Physics, vol. 21, No 6 June 1953.*

[35] W. Hastings, "Monte Carlo Sampling Methods Using Markov Chains and Their Applications." *Biometrika vol. 57, No. 1, pp. 97–109, April, 1970.*

[36] S. Brookseiman, "Markov Chain Monte Carlo Method and Its Application." *Journal of the Royal Statistical Society. Series D (The Statistician), Vol. 47, No. 1 , pp. 69–100, 1998.*

[37] I. Witten and E. Frank, "Data Mining: Practical Machine Learning Tools and Techniques With Java Implementations." *Morgan Kaufmann 2000.*

[38] J. R. Quinlan, "C4.5: Programs for Machine Learning." *Morgan Kaufmann Publishers, 1993.*

[39] C. Cortes and V. Vapnik, "Support-Vector Networks." *Machine Learning, vol 20, pp. 273–297, September 1995.*